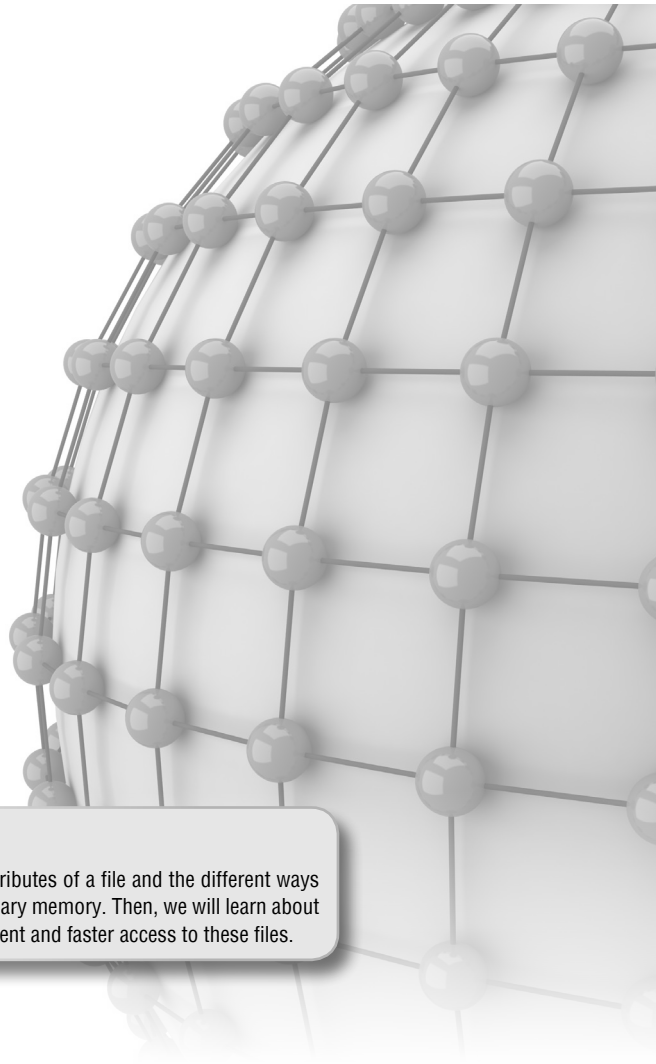
CHAPTER 16

Files and Their Organization



LEARNING OBJECTIVE

In this chapter, we will discuss the basic attributes of a file and the different ways in which files can be organized in the secondary memory. Then, we will learn about different indexing strategies that allow efficient and faster access to these files.

16.1 INTRODUCTION

Nowadays, most organizations use data collection applications which collect large amounts of data in one form or other. For example, when we seek admission in a college, a lot of data such as our name, address, phone number, the course in which we want to seek admission, aggregate of marks obtained in the last examination, and so on, are collected. Similarly, to open a bank account, we need to provide a lot of input. All these data were traditionally stored on paper documents, but handling these documents had always been a chaotic and difficult task.

Similarly, scientific experiments and satellites also generate enormous amounts of data. Therefore, in order to efficiently analyse all the data that has been collected from different sources, it has become a necessity to store the data in computers in the form of files.

In computer terminology, a file is a block of useful data which is available to a computer program and is usually stored on a persistent storage medium. Storing a file on a persistent storage medium like hard disk ensures the availability of the file for future use. These days, files stored on computers are a good alternative to paper documents that were once stored in offices and libraries.

16.2 DATA HIERARCHY

Every file contains data which can be organized in a hierarchy to present a systematic organization. The data hierarchy includes data items such as fields, records, files, and database. These terms are defined below.

- A *data field* is an elementary unit that stores a single fact. A data field is usually characterized by its type and size. For example, student's name is a data field that stores the name of students. This field is of type *character* and its size can be set to a maximum of 20 or 30 characters depending on the requirement.
- A *record* is a collection of related data fields which is seen as a single unit from the application point of view. For example, the student's record may contain data fields such as name, address, phone number, roll number, marks obtained, and so on.
- A *file* is a collection of related records. For example, if there are 60 students in a class, then there are 60 records. All these related records are stored in a file. Similarly, we can have a file of all the employees working in an organization, a file of all the customers of a company, a file of all the suppliers, so on and so forth.
- A *directory* stores information of related files. A directory organizes information so that users can find it easily. For example, consider Fig. 16.1 that shows how multiple related files are stored in a student directoy.

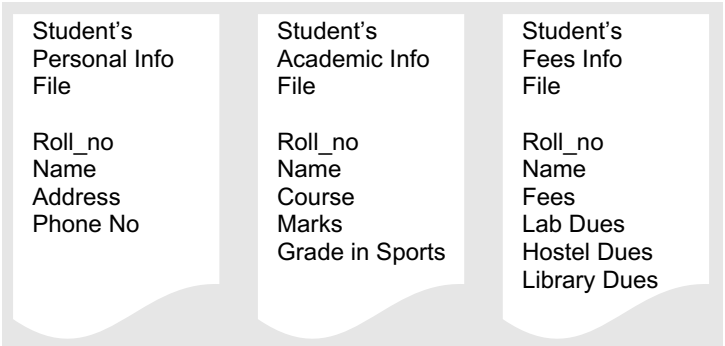


Figure 16.1 Student directory

16.3 FILE ATTRIBUTES

Every file in a computer system is stored in a directory. Each file has a list of attributes associated with it that gives the operating system and the application software information about the file and how it is intended to be used.

A software program which needs to access a file looks up the directory entry to discern the attributes of that file. For example, if a user attempts to write to a file that has been marked as a read-only file, then the program prints an appropriate message to notify the user that he is trying to write to a file that is meant only for reading.

Similarly, there is an attribute called *hidden*. When you execute the `DIR` command in DOS, then the files whose hidden attribute is set will not be displayed. These attributes are explained in this section.

File name It is a string of characters that stores the name of a file. File naming conventions vary from one operating system to the other.

File position It is a pointer that points to the position at which the next read/write operation will be performed.

File structure It indicates whether the file is a text file or a binary file. In the text file, the numbers (integer or floating point) are stored as a string of characters. A binary file, on the other hand, stores numbers in the same way as they are represented in the main memory.

File Access Method

It indicates whether the records in a file can be accessed sequentially or randomly. In sequential access mode, records are read one by one. That is, if 60 records of students are stored in the STUDENT file, then to read the record of 39th student, you have to go through the record of the first 38 students. However, in random access, records can be accessed in any order.

Attributes Flag

A file can have six additional attributes attached to it. These attributes are usually stored in a single byte, with each bit representing a specific attribute. If a particular bit is set to '1' then this means that the corresponding attribute is turned on. Table 16.1 shows the list of attributes and their position in the attribute flag or attribute byte.

Table 16.1 Attribute flag

Attribute	Attribute Byte
Read-Only	00000001
Hidden	00000010
System	00000100
Volume Label	00001000
Directory	00010000
Archive	00100000

If a system file is set as hidden and read-only, then its attribute byte can be given as 00000111. We will discuss all these attributes here in this section. Note that the directory is treated as a special file in the operating system. So, all these attributes are applicable to files as well as to directories.

Read-only A file marked as read-only cannot be deleted or modified. For example, if an attempt is made to either delete or modify a read-only file, then a message 'access denied' is displayed on the screen.

Hidden A file marked as hidden is not displayed in the directory listing.

System A file marked as a system file indicates that it is an important file used by the system and should not be altered or removed from the disk. In essence, it is like a 'more serious' read-only flag.

Volume Label Every disk volume is assigned a label for identification. The label can be assigned at the time of formatting the disk or later through various tools such as the DOS command LABEL.

Directory In directory listing, the files and sub-directories of the current directory are differentiated by a directory-bit. This means that the files that have the directory-bit turned on are actually sub-directories containing one or more files.

Archive The archive bit is used as a communication link between programs that modify files and those that are used for backing up files. Most backup programs allow the user to do an incremental backup. Incremental backup selects only those files for backup which have been modified since the last backup.

When the backup program takes the backup of a file, or in other words, when the program archives the file, it clears the archive bit (sets it to zero). Subsequently, if any program modifies the file, it turns on the archive bit (sets it to 1). Thus, whenever the backup program is run, it checks the archive bit to know whether the file has been modified since its last run. The backup program will archive only those files which were modified.

16.4 TEXT AND BINARY FILES

A *text file*, also known as a flat file or an ASCII file, is structured as a sequence of lines of alphabet, numerals, special characters, etc. However, the data in a text file, whether numeric or non-numeric, is stored using its corresponding ASCII code. The end of a text file is often denoted by placing a special character, called an end-of-file marker, after the last line in the text file.

A *binary file* contains any type of data encoded in binary form for computer storage and processing purposes. A binary file can contain text that is not broken up into lines. A binary file stores data in a format that is similar to the format in which the data is stored in the main memory.

Therefore, a binary file is not readable by humans and it is up to the program reading the file to make sense of the data that is stored in the binary file and convert it into something meaningful (e.g., a fixed length of record).

Binary files contain formatting information that only certain applications or processors can understand. It is possible for humans to read text files which contain only ASCII text, while binary files must be run on an appropriate software or processor so that the software or processor can transform the data in order to make it readable. For example, only Microsoft Word can interpret the formatting information in a Word document.

Although text files can be manipulated by any text editor, they do not provide efficient storage. In contrast, binary files provide efficient storage of data, but they can be read only through an appropriate program.

16.5 BASIC FILE OPERATIONS

The basic operations that can be performed on a file are given in Fig. 16.2.

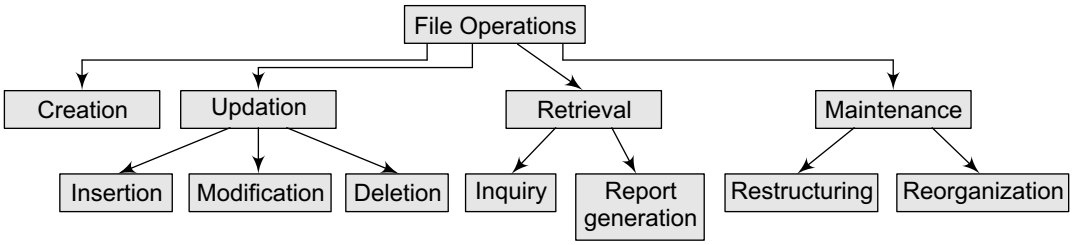


Figure 16.2 File operations

Creating a File

A file is created by specifying its name and mode. Then the file is opened for writing records that are read from an input device. Once all the records have been written into the file, the file is closed. The file is now available for future read/write operations by any program that has been designed to use it in some way or the other.

Updating a File

Updating a file means changing the contents of the file to reflect a current picture of reality. A file can be updated in the following ways:

- Inserting a new record in the file. For example, if a new student joins the course, we need to add his record to the STUDENT file.
- Deleting an existing record. For example, if a student quits a course in the middle of the session, his record has to be deleted from the STUDENT file.
- Modifying an existing record. For example, if the name of a student was spelt incorrectly, then correcting the name will be a modification of the existing record.

Retrieving from a File

It means extracting useful data from a given file. Information can be retrieved from a file either for an inquiry or for report generation. An inquiry for some data retrieves low volume of data, while report generation may retrieve a large volume of data from the file.

Maintaining a File

It involves restructuring or re-organizing the file to improve the performance of the programs that access this file. Restructuring a file keeps the file organization unchanged and changes only the structural aspects of the file (for example, changing the field width or adding/deleting fields). On

the other hand, file reorganization may involve changing the entire organization of the file. We will discuss file organization in detail in the next section.

16.6 FILE ORGANIZATION

We know that a file is a collection of related records. The main issue in file management is the way in which the records are organized inside the file because it has a significant effect on the system performance. Organization of records means the *logical* arrangement of records in the file and not the physical layout of the file as stored on a storage media.

Since choosing an appropriate file organization is a design decision, it must be done keeping the priority of achieving good performance with respect to the most likely usage of the file. Therefore, the following considerations should be kept in mind before selecting an appropriate file organization method:

- Rapid access to one or more records
- Ease of inserting/updating/deleting one or more records without disrupting the speed of accessing record(s)
- Efficient storage of records
- Using redundancy to ensure data integrity

Although one may find that these requirements are in contradiction with each other, it is the designer's job to find a good compromise among them to get an adequate solution for the problem at hand. For example, the ease of addition of records can be compromised to get fast access to data.

In this section, we will discuss some of the techniques that are commonly used for file organization.

16.6.1 Sequential Organization

A sequentially organized file stores the records in the order in which they were entered. That is, the first record that was entered is written as the first record in the file, the second record entered is written as the second record in the file, and so on. As a result, new records are added only at the end of the file.

Sequential files can be read only sequentially, starting with the first record in the file. Sequential file organization is the most basic way to organize a large collection of records in a file. Figure 16.3 shows `n` records numbered from `0` to `n-1` stored in a sequential file.

Once we store the records in a file, we cannot make any changes to the records. We cannot even delete the records from a sequential file. In case we need to delete or update one or more records, we have to replace the records by creating a new file.

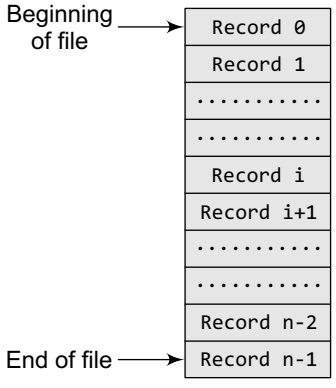


Figure 16.3 Sequential file organization

In sequential file organization, all the records have the same size and the same field format, and every field has a fixed size. The records are sorted based on the value of one field or a combination of two or more fields. This field is known as the *key*. Each key uniquely identifies a record in a file. Thus, every record has a different value for the key field. Records can be sorted in either ascending or descending order.

Sequential files are generally used to generate reports or to perform sequential reading of large amount of data which some programs need to do such as payroll processing of all the employees of an organization. Sequential files can be easily stored on both disks and tapes. Table 16.2 summarizes the features, advantages, and disadvantages of sequential file organization.

Table 16.2 Sequential file organization

Features	Advantages	Disadvantages
• Records are written in the order in which they are entered • Records are read and written sequentially • Deletion or updation of one or more records calls for replacing the original file with a new file that contains the desired changes • Records have the same size and the same field format • Records are sorted on a key value • Generally used for report generation or sequential reading	• Simple and easy to handle • No extra overheads involved • Sequential files can be stored on magnetic disks as well as magnetic tapes • Well suited for batch-oriented applications	• Records can be read only sequentially. If i^{th} record has to be read, then all the i–1 records must be read • Does not support update operation. A new file has to be created and the original file has to be replaced with the new file that contains the desired changes • Cannot be used for interactive applications

16.6.2 Relative File Organization

Relative file organization provides an effective way to access individual records directly. In a relative file organization, records are ordered by their *relative key.* It means the record number represents the location of the record relative to the beginning of the file. The record numbers range from `0` to `n-1`, where `n` is the number of records in the file. For example, the record with record number `0` is the first record in the file. The records in a relative file are of fixed length.

Therefore, in relative files, records are organized in ascending *relative record number.* A relative file can be thought of as a single dimension table stored on a disk, in which the relative record number is the index into the table. Relative files can be used for both random as well as sequential access. For sequential access, records are simply read one after another.

Relative files provide support for only one key, that is, the relative record number. This key must be numeric and must take a value between 0 and the current highest relative record number –1. This means that enough space must be allocated for the file to contain the records with relative record numbers between 0 and the highest record number –1. For example, if the highest relative record number is 1,000, then space must be allocated to store 1,000 records in the file.

Figure 16.4 shows a schematic representation of a relative file which has been allocated enough space to store 100 records. Although it has space to accommodate 100 records, not all the locations are occupied. The locations marked as `FREE` are yet to store records in them. Therefore, every location in the table either stores a record or is marked as `FREE`.

Relative record number	Records stored in memory
0	Record 0
1	Record 1
2	FREE
3	FREE
4	Record 4
.....	.....
98	FREE
99	Record 99

Figure 16.4 Relative file organization

Relative file organization provides random access by directly jumping to the record which has to be accessed. If the records are of fixed length and we know the base address of the file and the length of the record, then any record `i` can be accessed using the following formula:

```
Address of ith record = base_address + (i-1) * record_length
```

Note that the base address of the file refers to the starting address of the file. We took `i-1` in the formula because record numbers start from 0 rather than 1.

Consider the base address of a file is 1000 and each record occupies 20 bytes, then the address of the 5^{th} record can be given

as:

```
1000 + (5–1) * 20
= 1000 + 80
= 1080
```

Table 16.3 summarizes the features, advantages, and disadvantages of relative file organization.

Table 16.3 Relative file organization

Features	Advantages	Disadvantages
• Provides an effective way to access individual records • The record number represents the location of the record relative to the beginning of the file • Records in a relative file are of fixed length • Relative files can be used for both random as well as sequential access • Every location in the table either stores a record or is marked as FREE	• Ease of processing • If the relative record number of the record that has to be accessed is known, then the record can be accessed instantaneously • Random access of records makes access to relative files fast • Allows deletions and updations in the same file • Provides random as well as sequential access of records with low overhead • New records can be easily added in the free locations based on the relative record number of the record to be inserted • Well suited for interactive applications	• Use of relative files is restricted to disk devices • Records can be of fixed length only • For random access of records, the relative record number must be known in advance

16.6.3 Indexed Sequential File Organization

Indexed sequential file organization stores data for fast retrieval. The records in an indexed sequential file are of fixed length and every record is uniquely identified by a key field. We maintain a table known as the *index table* which stores the record number and the address of all the records. That is for every file, we have an index table. This type of file organization is called as indexed sequential file organization because physically the records may be stored anywhere, but the index table stores the address of those records.

The `ith` entry in the index table points to the `ith` record of the file. Initially, when the file is created, each entry in the index table contains `NULL`. When the `ith` record of the file is written, free space is obtained from the free space manager and its address is stored in the `ith` location of the index table.

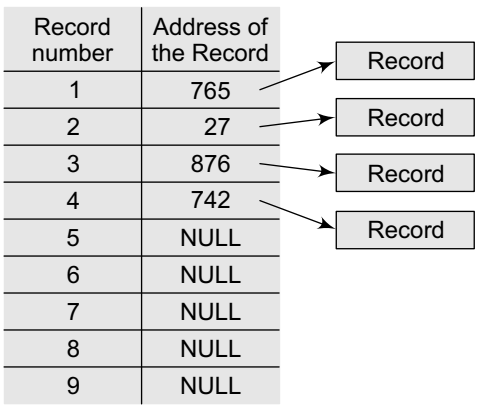


Figure 16.5 Indexed sequential file organization

Now, if one has to read the `4th` record, then there is no need to access the first three records. Address of the `4th` record can be obtained from the index table and the record can be straightaway read from the specified address (`742`, in our example). Conceptually, the index sequential file organization can be visualized as shown in Fig. 16.5.

An indexed sequential file uses the concept of both sequential as well as relative files. While the index table is read sequentially to find the address of the desired record, a direct access is made to the address of the specified record in order to access it randomly.

Indexed sequential files perform well in situations where sequential access as well as random access is made to

the data. Indexed sequential files can be stored only on devices that support random access, for example, magnetic disks.

For example, take an example of a college where the details of students are stored in an indexed sequential file. This file can be accessed in two ways:

- *Sequentially*—to print the aggregate marks obtained by each student in a particular course or
- *Randomly*—to modify the name of a particular student.

Table 16.4 summarizes the features, advantages, and disadvantages of indexed sequential file organization.

Table 16.4 Indexed sequential file organization

Features	Advantages	Disadvantages
• Provides fast data retrieval • Records are of fixed length • Index table stores the address of the records in the file • The ith entry in the index table points to the ith record of the file • While the index table is read sequentially to find the address of the desired record, a direct access is made to the address of the specified record in order to access it randomly • Indexed sequential files perform well in situations where sequential access as well as random access is made to the data	• The key improvement is that the indices are small and can be searched quickly, allowing the database to access only the records it needs • Supports applications that require both batch and interactive processing • Records can be accessed sequentially as well as randomly • Updates the records in the same file	• Indexed sequential files can be stored only on disks • Needs extra space and overhead to store indices • Handling these files is more complicated than handling sequential files • Supports only fixed length records

16.7 INDEXING

An index for a file can be compared with a catalogue in a library. Like a library has card catalogues based on authors, subjects, or titles, a file can also have one or more indices.

Indexed sequential files are very efficient to use, but in real-world applications, these files are very large and a single file may contain millions of records. Therefore, in such situations, we require a more sophisticated indexing technique. There are several indexing techniques and each technique works well for a particular application. For a particular situation at hand, we analyse the indexing technique based on factors such as access type, access time, insertion time, deletion time, and space overhead involved. There are two kinds of indices:

- *Ordered indices* that are sorted based on one or more key values
- *Hash indices* that are based on the values generated by applying a *hash function*

16.7.1 Ordered Indices

Indices are used to provide fast random access to records. As stated above, a file may have multiple indices based on different key fields. An index of a file may be a primary index or a secondary index.

Primary Index

In a sequentially ordered file, the index whose search key specifies the sequential order of the file is defined as the primary index. For example, suppose records of students are stored in a STUDENT file in a sequential order starting from roll number 1 to roll number 60. Now, if we want to search

a record for, say, roll number 10, then the student's roll number is the primary index. Indexed sequential files are a common example where a primary index is associated with the file.

Secondary Index

An index whose search key specifies an order different from the sequential order of the file is called as the secondary index. For example, if the record of a student is searched by his name, then the name is a secondary index. Secondary indices are used to improve the performance of queries on non-primary keys.

16.7.2 Dense and Sparse Indices

In a dense index, the index table stores the address of every record in the file. However, in a sparse index, the index table stores the address of only some of the records in the file. Although sparse indices are easy to fit in the main memory, a dense index would be more efficient to use than a sparse index if it fits in the memory. Figure 16.6 shows a dense index and a sparse index for an indexed sequential file.

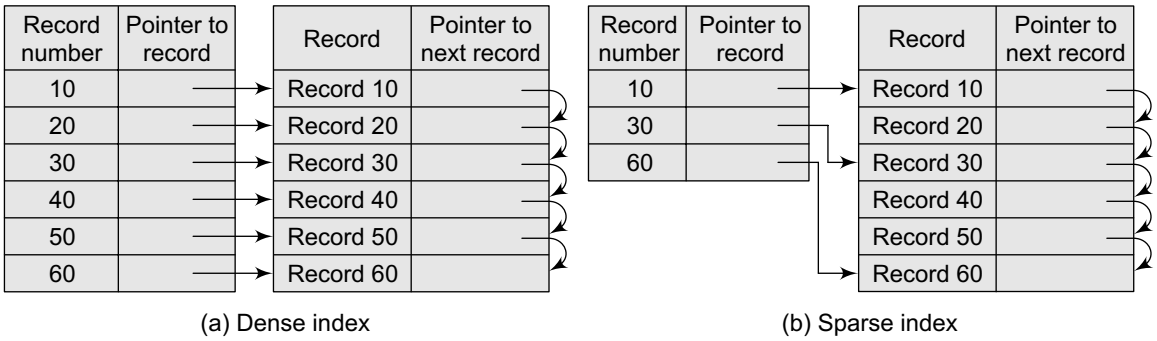


Figure 16.6 Dense index and sparse index

Note that the records need not be stored in consecutive memory locations. The pointer to the next record stores the address of the next record.

By looking at the dense index, it can be concluded directly whether the record exists in the file or not. This is not the case in a sparse index. In a sparse index, to locate a record, we first find an entry in the index table with the largest search key value that is either less than or equal to the search key value of the desired record. Then, we start at that record pointed to by that entry in the index table and then proceed searching the record using the sequential pointers in the file, until the desired record is obtained. For example, if we need to access record number 40, then record number 30 is the largest key value that is less than 40. So jump to the record pointed by record number 30 and move along the sequential pointer to reach record number 40.

Thus we see that sparse index takes more time to find a record with the given key. Dense indices are faster to use, while sparse indices require less space and impose less maintenance for insertions and deletions.

16.7.3 Cylinder Surface Indexing

Cylinder surface indexing is a very simple technique used only for the primary key index of a sequentially ordered file. In a sequentially ordered file, the records are stored sequentially in the increasing order of the primary key. The index file will contain two fields—cylinder index and several surface indices. Generally, there are multiple cylinders, and each cylinder has multiple surfaces. If the file needs `m` cylinders for storage then the cylinder index will contain `m` entries.

Each cylinder will have an entry corresponding to the largest key value into that cylinder. If the disk has `n` usable surfaces, then each of the surface indices will have `n` entries. Therefore, the `ith` entry in the surface index for cylinder `j` is the largest key value on the `jth` track of the `ith` surface. Hence, the total number of surface index entries is `m.n`. The physical and logical organization of disk is shown in Fig. 16.7.

Note The number of cylinders in a disk is only a few hundred and the cylinder index occupies only one track.

When a record with a particular key value has to be searched, then the following steps are performed:

- First the cylinder index of the file is read into memory.
- Second, the cylinder index is searched to determine which cylinder holds the desired record. For this, either the binary search technique can be used or the cylinder index can be made to store an array of pointers to the starting of individual key values. In either case the search will take `O (log m)` time.
- After the cylinder index is searched, appropriate cylinder is determined.
- Depending on the cylinder, the surface index corresponding to the cylinder is then retrieved from the disk.
- Since the number of surfaces on a disk is very small, linear search can be used to determine surface index of the record.
- Once the cylinder and the surface are determined, the corresponding track is read and searched for the record with the desired key.

Hence, the total number of disk accesses is three—first, for accessing the cylinder index, second for accessing the surface index, and third for getting the track address. However, if track sizes are very large then it may not be a good idea to read the whole track at once. In such situations, we can also include sector addresses. But this would add an extra level of indexing and, therefore, the number of accesses needed to retrieve a record will then become four. In addition to this, when the file extends over several disks, a disk index will also be added.

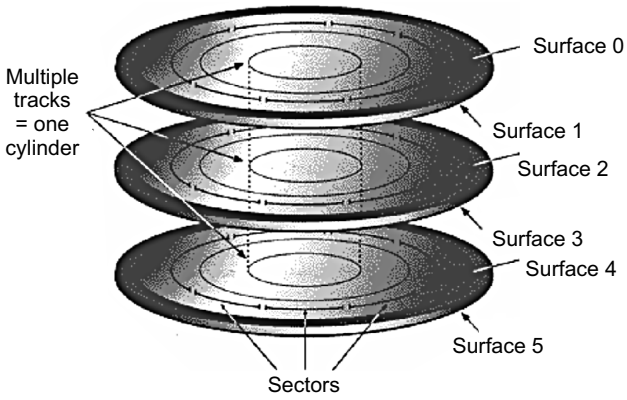


Figure 16.7 Physical and logical organization of disk

The cylinder surface indexing method of maintaining a file and index is referred to as Indexed Sequential Access Method (ISAM). This technique is the most popular and simplest file organization in use for single key values. But with files that contain multiple keys, it is not possible to use this index organization for the remaining keys.

16.7.4 Multi-level Indices

In real-world applications, we have very large files that may contain millions of records. For such files, a simple indexing technique will not suffice. In such a situation, we use multi-level indices. To understand this concept, consider a file that has 10,000 records. If we use simple indexing, then we need an index table that can contain at least 10,000 entries to point to 10,000 records. If

each entry in the index table occupies 4 bytes, then we need an index table of 4×10000 bytes = 40000 bytes. Finding such a big space consecutively is not always easy. So, a better scheme is to index the index table.

Figure 16.8 shows a two-level multi-indexing. We can continue further by having a three-level indexing and so on. But practically, we use two-level indexing. Note that two and higher-level indexing must always be sparse, otherwise multi-level indexing will lose its effectiveness. In the figure, the main index table stores pointers to three inner index tables. The inner index tables are sparse index tables that in turn store pointers to the records.

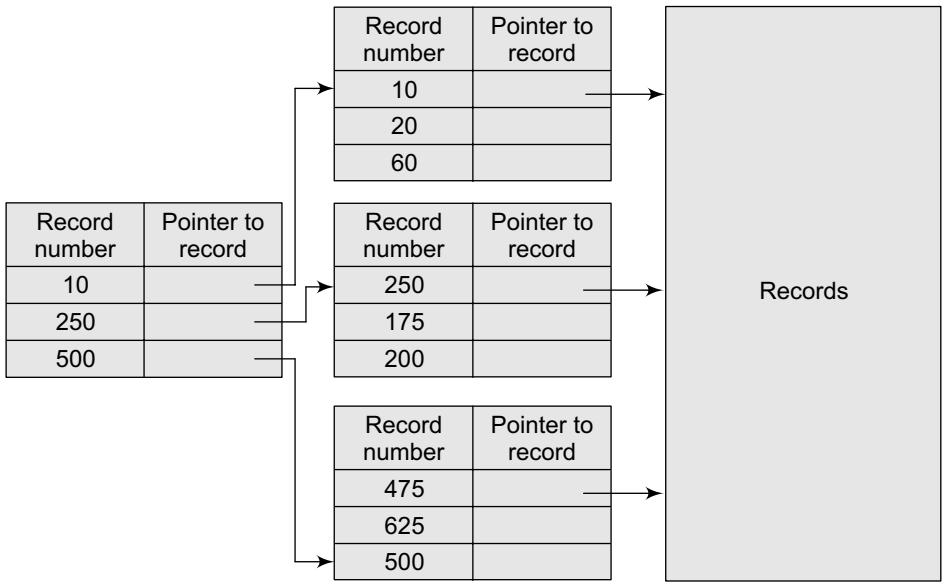


Figure 16.8 Multi-level indices

16.7.5 Inverted Indices

Inverted files are commonly used in document retrieval systems for large textual databases. An inverted file reorganizes the structure of an existing data file in order to provide fast access to all records having one field falling within the set limits.

For example, inverted files are widely used by bibliographic databases that may store author names, title words, journal names, etc. When a term or keyword specified in the inverted file is identified, the record number is given and a set of records corresponding to the search criteria are created.

Thus, for each keyword, an inverted file contains an inverted list that stores a list of pointers to all occurrences of that term in the main text. Therefore, given a keyword, the addresses of all the documents containing that keyword can easily be located.

There are two main variants of inverted indices:

- A record-level inverted index (also known as *inverted file index* or *inverted file*) stores a list of references to documents for each word
- A word-level inverted index (also known as *full inverted index* or *inverted list*) in addition to a list of references to documents for each word also contains the positions of each word within a document. Although this technique needs more time and space, it offers more functionality (like phrase searches)

Therefore, the inverted file system consists of an index file in addition to a document file (also known as *text file*). It is this index file that contains all the keywords which may be used as search terms. For each keyword, an address or reference to each location in the document where that word occurs is stored. There is no restriction on the number of pointers associated with each word.

For efficiently retrieving a word from the index file, the keywords are sorted in a specific order (usually alphabetically).

However, the main drawback of this structure is that when new words are added to the documents or text files, the whole file must be reorganized. Therefore, a better alternative is to use B-trees.

16.7.6 B-Tree Indices

A database is defined as a collection of data organized in a fashion that facilitates updating, retrieving, and managing the data (that may include any item, such as names, addresses, pictures, and numbers). Most organizations maintain databases for their business operations. For example, an airline reservation system maintains a database of flights, customers, and tickets issued. A university maintains a database of all its students. These real-world databases may contain millions of records that may occupy gigabytes of storage space.

For a database to be useful, it must support fast retrieval and storage of data. Since it is impractical to maintain the entire database in the memory, B-trees are used to index the data in order to provide fast access.

For example, searching a value in an un-indexed and unsorted database containing `n` key values may take a running time of `O(n)` in the worst case, but if the same database is indexed with a B-tree, the search operation will run in `O(log n)` time.

Majority of the database management systems use the B-tree index technique as the default indexing method. This technique supersedes other techniques of creating indices, mainly due to its data retrieval speed, ease of maintenance, and simplicity. Figure 16.9 shows a B-tree index.

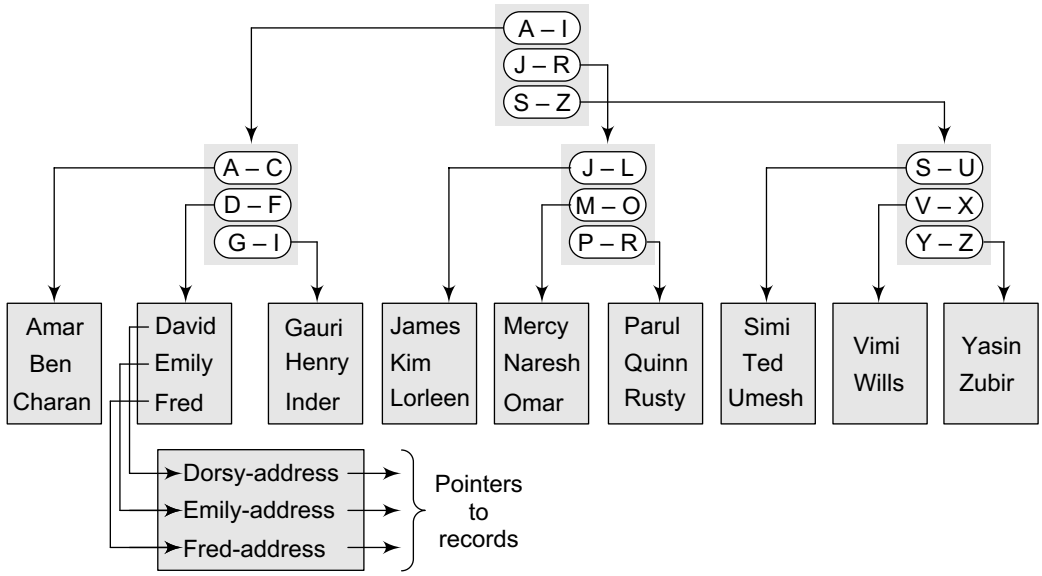


Figure 16.9 B-tree index

It forms a tree structure with the root at the top. The index consists of a B-tree (balanced tree) structure based on the values of the indexed column. In this example, the indexed column is *name* and the B-tree is created using all the existing names that are the values of the indexed column. The upper blocks of the tree contain index data pointing to the next lower block, thus forming a hierarchical structure. The lowest level blocks, also known as leaf blocks, contain pointers to the data rows stored in the table.

If a table has a column that has many unique values, then the selectivity of that column is said to be high. B-tree indices are most suitable for highly selective columns, but it causes a sharp increase in the size when the indices contain concatenation of multiple columns.

The B-tree structure has the following advantages:

- Since the leaf nodes of a B-tree are at the same depth, retrieval of any record from anywhere in the index takes approximately the same time.
- B-trees improve the performance of a wide range of queries that either search a value having an exact match or for a value within specified range.
- B-trees provide fast and efficient algorithms to insert, update, and delete records that maintain the key order.
- B-trees perform well for small as well as large tables. Their performance does not degrade as the size of a table grows.
- B-trees optimize costly disk access.

16.7.7 Hashed Indices

In the last chapter, we discussed hashing in detail. The same concept of hashing can be used to create hashed indices.

So far, we have studied that hashing is used to compute the address of a record by using a hash function on the search key value. If at any point of time, the hashed values map to the same address, then collision occurs and schemes to resolve these collisions are applied to generate a new address.

Choosing a good hash function is critical to the success of this technique. By a good hash function, we mean two things. First, a good hash function, irrespective of the number of search keys, gives an average-case lookup that is a small constant. Second, the function distributes records uniformly and randomly among the buckets, where a bucket is defined as a unit of one or more records (typically a disk block). Correspondingly, the worst hash function is one that maps all the keys to the same bucket.

However, the drawback of using hashed indices includes:

- Though the number of buckets is fixed, the number of files may grow with time.
- If the number of buckets is too large, storage space is wasted.
- If the number of buckets is too small, there may be too many collisions.

It is recommended to set the number of buckets to twice the number of the search key values in the file. This gives a good space–performance tradeoff.

A hashed file organization uses hashed indices. Hashing is used to calculate the address of disk block where the desired record is stored. If K is the set of all search key values and B is the set of all bucket addresses, then a hash function H maps K to B.

We can perform the following operations in a hashed file organization.

Insertion

To insert a record that has k_i as its search value, use the hash function $h(k_i)$ to compute the address of the bucket for that record. If the bucket is free, store the record else use chaining to store the record.

Search

To search a record having the key value k_i, use $h(k_i)$ to compute the address of the bucket where the record is stored. The bucket may contain one or several records, so check for every record in the bucket (by comparing k_i with the key of every record) to finally retrieve the desired record with the given key value.

Deletion

To delete a record with key value k_i, use $h(k_i)$ to compute the address of the bucket where the record is stored. The bucket may contain one or several records so check for every record in the bucket (by comparing k_i with the key of every record). Then delete the record as we delete a node from a linear linked list. We have already studied how to delete a record from a chained hash table in Chapter 15.

Note that in a hashed file organization, the secondary indices need to be organized using hashing.

Points to Remember

- A file is a block of useful information which is available to a computer program and is usually stored on a persistent storage medium.
- Every file contains data. This data can be organized in a hierarchy to present a systematic organization. The data hierarchy includes data items such as fields, records, files, and database.
- A data field is an elementary unit that stores a single fact. A record is a collection of related data fields which is seen as a single unit from the application point of view. A file is a collection of related records. A directory is a collection of related files.
- A database is defined as a collection of data organized in a fashion that facilitates updating, retrieving, and managing the data.
- There are two types of computer files—text files and binary files. A text file is structured as a sequence of lines of alphabet, numbers, special characters, etc. However, the data in a text file is stored using its corresponding ASCII code. Whereas in binary files, the data is stored in binary form, i.e., in the format it is stored in the memory.
- Each file has a list of attributes associated with it which can have one of two states—*on* or *off*. These attributes are: read-only, hidden, system, volume label, archive, and directory.
- A file marked as read-only cannot be deleted or modified.
- A hidden file is not displayed in the directory listing.
- A system file is used by the system and should not be altered or removed from the disk.
- The archive bit is useful for communication between programs that modify files and programs that are used for backing up files.
- A file that has the directory bit turned on is actually a sub-directory containing one or more files.
- File organization means the logical arrangement of records in the file. Files can be organized as sequential, relative, or index sequential.
- A sequentially organized file stores records in the order in which they were entered.
- In relative file organization, records in a file are ordered by their relative key. Relative files can be used for both random access as well as sequential access of data.
- In an indexed sequential file, every record is uniquely identified by a key field. We maintain a table known as the index table that stores record number and the address of the record in the file.
- There are several indexing techniques, and each technique works well for a particular application.
- In a dense index, index table stores the address of every record in the file. However, in a sparse index, index table stores address of only some of the records in the file.
- Cylinder surface indexing is a very simple technique which is used only for the primary key index of a sequentially ordered file.
- In multi-level indexing, we can create an index to the index itself. The original index is called the first-level index and the index to the index is called the second-level index.

- Inverted files are frequently used indexing technique in document retrieval systems for large textual databases. An inverted file reorganizes the structure of an existing data file in order to provide fast access to all records having one field falling within set limits.
- Majority of the database management systems use B-tree indexing technique. The index consists of a hierarchical structure with upper blocks containing indices pointing to the lower blocks and lowest level blocks containing pointers to the data records.
- Hashed file organization uses hashed indices. Hashing is used to calculate the address of disk block where the desired record is stored. If `K` is the set of all search key values and `B` is the set of bucket addresses, then a hash function `H` maps `K` to `B`.

EXERCISES

Review Questions

1. Why do we need files?
2. Explain the terms field, record, file organization, key, and index.
3. Define file. Explain all the file attributes.
4. How is archive attribute useful?
5. Differentiate between a binary file and a text file.
6. Explain the basic file operations.
7. What do you understand by the term file organization? Briefly summarize the different file organizations that are widely used today.
8. Write a brief note on indexing.
9. Differentiate between sparse index and dense index.
10. Explain the significance of multi-level indexing with an appropriate example.
11. What are inverted files? Why are they needed?
12. Give the merits and demerits of a B-tree index.

Multiple-choice Questions

1. Which of the following flags is cleared when a file is backed up?
   (a) Read-only
   (b) System
   (c) Hidden
   (d) Archive
2. Which is an important file used by the system and should not be altered or removed from the disk?
   (a) Hidden file
   (b) Archived file
   (c) System file
   (d) Read-only file
3. The data hierarchy can be given as
   (a) Fields, records, files and database
   (b) Records, files, fields and database
   (c) Database, files, records and fields
   (d) Fields, records, database, and files
4. Which of the following indexing techniques is used in document retrieval systems for large databases?
   (a) Inverted index
   (b) Multi-level indices
   (c) Hashed indices
   (d) B-tree index

True or False

1. When a backup program archives the file, it sets the archive bit to one.
2. In a text file, data is stored using ASCII codes.
3. A binary file is more efficient than a text file.
4. Maintenance of a file involves re-structuring or re-organization of the file.
5. Relative files can be used for both random access of data as well as sequential access.
6. In a sparse index, index table stores the address of every record in the file.
7. Higher level indexing must always be sparse.
8. B-tree indices are most suitable for highly selective columns.

Fill in the Blanks

1. _____ is a block of useful information.
2. A data field is usually characterized by its _____ and _____.
3. _____ is a collection of related data fields.

4. _____ is a pointer that points to the position at which next read/write operation will be performed.
5. _____ indicates whether the file is a text file or a binary file.
6. Index table stores _____ and _____ of the record in the file.
7. In a sequentially ordered file the index whose search key specifies the sequential order of the file is defined as the _____ index.
8. _____ files are frequently used indexing technique in document retrieval systems for large textual databases.
9. _____ is a collection of data organized in a fashion that facilitates updating, retrieving, and managing the data.